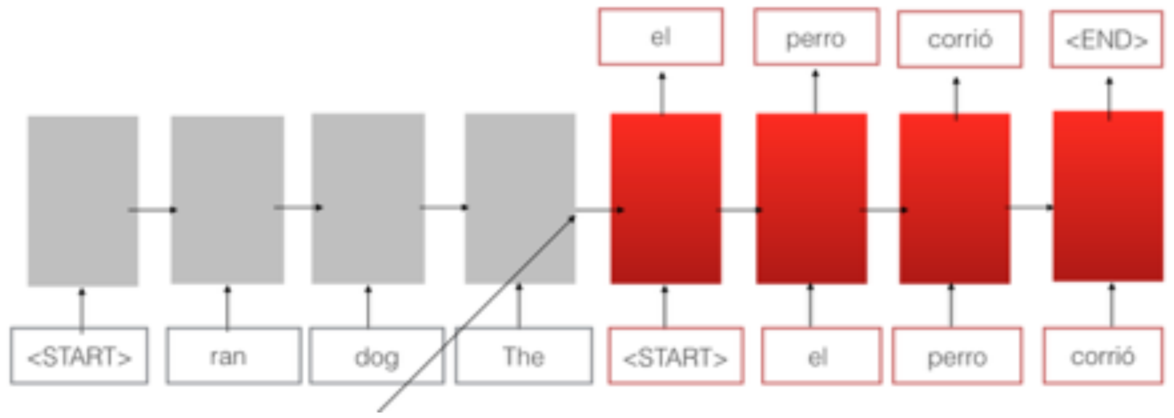
ZOPH_RNN

A GPU Recurrent Neural Network Toolkit Tutorial

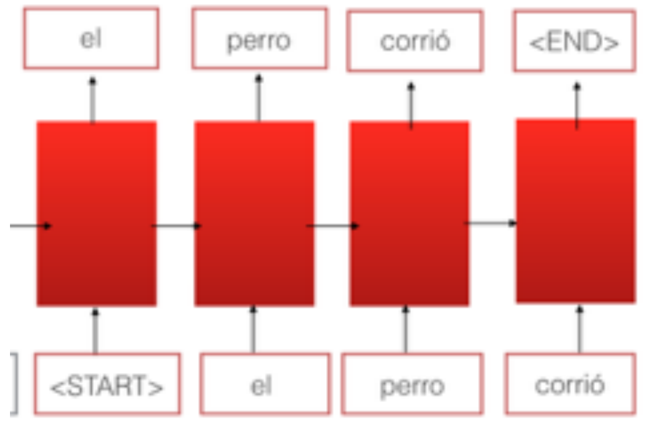
Barret Zoph

Brief RNN Overview

- The RNN architectures in my toolkit have two different types of models
  - Learn functions mapping sequences to sequences (like machine translation)



  - Learn functions learning single sequences (like language modeling)



Requirements

- The code can be compiled on any operating system, but the executable (RNN_MODEL) is precompiled for 64 bit linux.

- To run the code you need CUDA 7.0 or higher.

- To compile use the following:

- nvcc -O3 -Xcompiler -fopenmp -I <YOUR PATH TO CUDA 7.0> -I <YOUR PATH TO BOOST> <PATH TO libboost_system.a >   <PATH TO libboost_filesystem.a > <PATH TO libboost_program_options.a> -I <YOUR PATH TO EIGEN> -std=c++11 -lcublas -lcurand main.cu

- Throughout the tutorial the % symbol denotes commands that can be run on the command line

Program Usage

- If you are ever confused run ./RNN_MODEL -h for the help menu! It will show you all the flags and commands for the program.

- There are 3 things main things you can do with the code:

  1. Train a model (either sequence like LM, or sequence to sequence like MT)

  2. Force-decode (get the per example probability plus perplexity of some dataset, either sequence or sequence to sequence models)

  3. Get the k-best paths using beam search for an already trained sequence to sequence model (note the code does not have this for the normal sequence model, as I am not sure why you would want this …)

Training Input Format

- What is the format for training a model using my code?
- Note that my code defaults to the sequence to sequence model, so just use the -s flag to train an only sequence model
- The <output file name> is the file where the trained neural network will be stored. It contains all of your model parameters, mappings for integerization and the model parameters. This is the only thing you need to keep around once you trained a model.
- The input files must have the tokens separated by spaces. This is the only requirement.
- %RNN_MODEL -t <source file name> <target file name> <output file name> /*For sequence to sequence model*/
    - RNN_MODEL -t english.txt spanish.txt model.nn   /*trains a MT system learning P(spanish | english) */
- %RNN_MODEL -s -t <target file name> <output file name>   /*for sequence model*/
    - RNN_MODEL -s -t english.txt model.nn   /*trains a LM learning P(english)*/

Force-Decode Format

- All you need is your neural network file after training! (model.nn in the previous examples)

- %RNN_MODEL -f <source file> <target file> <neural network file> <output file> /*for sequence to sequence model*/

    - %RNN_MODEL -f english_test.txt spanish_test.txt model.nn output.txt   /*get P(spanish_test.txt | english_test.txt)*/

- %RNN_MODEL -s -f <target file> <neural network file> <output file> /*for sequence model*/

    - %RNN_MODEL -s -f english_test.txt model.nn output.txt   /*get P(english_test.txt)*/

Kbest Path Format

- All you need is your neural network file after training! (model.nn in the previous examples)
- kbest does not exist for the sequence model
- %RNN_MODEL -k <how many paths> <source file> <neural network file> <output file> /*for sequence to sequence model*/
- %RNN_MODEL -k 10 english_test.txt model.nn output.txt /*get the 10 best paths using beam decoding with the input file being english_test.txt*/

Different Loss Functions

- The standard default loss function is maximum likelihood.
- The loss function can be switched to noise contrastive estimation with the - - NCE flag (NCE uses a unigram distribution under the hood).
- The NCE loss function shares all the noise samples across a mini batch so dense operations can be done on the GPU. This was found to not degrade performance.
- %RNN_MODEL -t <source file name> <target file name> <output file name> —NCE <number of noise samples> /*For sequence to sequence model with NCE*/
    - RNN_MODEL -t english.txt spanish.txt model.nn —NCE 100 /*trains a MT system learning P(spanish | english) with NCE using 100 noise samples */
- %RNN_MODEL -s -t <target file name> <output file name>   /*for sequence model*/
    - RNN_MODEL -s -t english.txt model.nn   —NCE 250   /*trains a LM learning P(english) using NCE with 250 noise samples*/

Setting the number of Layers

- The flag (-N <number of layers>) will set the number of LSTM layers that are in the sequence or sequence to sequence model.
    - %RNN_MODEL -t english.txt spanish.txt model.nn -N 4 /*trains a MT system learning P(spanish | english) with 4 LSTM layers*/

Multi GPU training

- To significantly speed up training you can train your models across multiple GPU's if they are connected together across the same PCI express bus

- The way to specify this is with the (-M <gpu indicies>) flag.

- The flag works as the following, if you have N GPUs connected together then they are numbered 0-9. Run the command to see the device numbers for your exact GPU setup:

    - %nvidia-smi

- Then you simply specify what layer you want on each GPU along with the loss function. The following command will put layer one on GPU 2, layer two on GPU 1, layer 3 on GPU 1, layer 4 on GPU 3 and the loss function (MLE or NCE) on GPU 6

    - %RNN_MODEL -t english.txt spanish.txt model.nn -N 4 -M 2 1 1 3 6

Learning Rate

- Do not make your learning rate too high! This can cause your loss function to start increasing. My code starts it at 0.7, as I found this works well for large datasets, but a value of 0.1-0.5 is standard.
- Conversely making your learning rate too small will cause your training to take a very long time to converge
- I have three flags in my code that can help deal with the learning rate.
- 1. (-l,—learning-rate), this will set the starting learning rate
  - RNN_MODEL -s -t english.txt model.nn -l 0.3   /*trains a model with initial learning rate of 0.3*/
- 2. (--halve-learning) This will get the perplexity of a user supplied dev set every half epoch and if the perplexity on the dev set increased last half epoch, I reduce the learning rate by an amount specified by the user
  - RNN_MODEL -t english.txt spanish.txt model.nn —adaptive-halve-lr english_dev.txt spanish_dev.txt   /*trains a model with the english_dev.txt and spanish_dev.txt being used to monitor learning rate halving*/
  - RNN_MODEL -s -t english.txt model.nn —adaptive-halve-lr english_dev.txt -A 0.7 /*trains a model with the english_dev.txt being used to monitor learning rate reduction. The -A 0.7 will now multiply the learning rate by 0.7 once the perplexity on the dev set increases as opposed to 0.5 by default*/
- 3. (—fixed-halve-learning) This will halve the learning rate every half epoch after a certain epoch has been reached.
  - RNN_MODEL -t english.txt spanish.txt model.nn —fixed-halve-lr 5 /*trains a model with the where the learning rate will be halved every epoch after the fifth epoch*/
  - RNN_MODEL -s -t english.txt model.nn —fixed-halve-lr 10 /*trains a model with the where the learning rate will be halved every epoch after the tenth epoch*/

Hidden state size

- Make the hidden state size large enough! I use 1000 for all the MT experiments.
- I have a flag to set the number of hidden states (-H,—hiddenstate-size)
    - RNN_GPU -s -t english.txt model.nn -H 200   /*trains a model with 200 hidden states*/
- I would recommend making the hiddenstate size 500+ for any non toy examples. But feel free to play around with it!
- If you make the hidden state size too big you make see a message (“GPU memory alloc failed…), this simply means there is not enough space on the GPU for the model, so reduce the hidden state size

Setting Vocab Sizes

- For sequence to sequence models the source vocabulary can be made arbitrarily big without it hurting the computational cost. The only thing to be careful about is running out of RAM on the GPU.

- By default it sets the source and target sizes to the number of unique tokens that appear in the training data when you train your neural network

- I have two flags to set the vocabulary sizes

    - 1. (-v,—source-vocab), this is only relevant for the sequence to sequence model. This sets the source vocal size

    - 2. (-V,—target-vocab), this sets the target vocal size

Decoding

- I set the standard beam-size to 12, but you can change this

- The flag (-b,—beam-size) will set it to whatever you want. Note increasing the beam size does decrease the speed

- If you notice that your neural network is outputting short sentences, play with the flag (-p,—penalty). This is by default zero and adds the amount specified to the unnormalized log prob of each word being decoded. The larger the value, the longer the sentences the model will prefer!

Gradient Clipping

- Be sure to clip the gradients of these models! The default is that each gradient matrix is clipped to have a norm <= 5. This can be changed in a few ways.

- The “-c <clip value>” flag will clip the gradient matrix to have a norm <= <clip value>

- The “-w <clip value>” flag will treat all the gradients as one big matrix and then make sure this norm <= <clip value>

- It is highly recommended that you have gradient clipping on as they gradients can get huge which makes the gradient descent be much more volatile

Dropout

- One way to prevent overfitting is to use dropout. The dropout implemented for these models is from the paper : http://arxiv.org/abs/1409.2329

- The flag (-d <dropout keep probability>) will make it so the probability of keeping a node is the <dropout keep probability>

Suggestions

- Find any bugs or have any suggestions to make the user input nicer/easier to use? Email me at zoph@isi.edu or barretzoph@gmail.com
- https://github.com/isi-nlp/Zoph_RNN